CS4102 - ALGORITHMS - SPRING 2022

DUE TUESDAY, MAY 3, 2022 AT 11:30 PM VIA GRADESCOPE

D PROGRAMMING

Collaboration Policy: You are encouraged to collaborate with up to 3 other students, but all work submitted must be your own *independently* written solution. List the computing ids of all of your collaborators in a comment at the top of your files. Do not share written notes, documents (including Google docs, Overleaf docs, discussion notes, PDFs), or code. Do not seek published or online solutions for any assignments. If you use any published or online resources (which may not include solutions) when completing this assignment, be sure to cite by naming the book etc. or listing a website's URL. Do not submit a solution that you are unable to explain orally to a member of the course staff. Any solutions that share similar text/code will be considered in breach of this policy. Please refer to the syllabus for a complete description of the collaboration policy.

Collaborators: List computing ids in comments at the top of your file

Sources: List URLs etc. in comments at the top of your file

Rather than tile an $2 \times n$ board, we would like to use dominoes to tile a pixel-art image, such as Figure 1. Unfortunately, not every such image can be tiled with dominoes. Write an algorithm that takes as input a black-and-white pixel-art image (with r rows and c columns) and determines whether or not that image's black pixels can be exactly tiled using 2-by-1 dominoes. The running time of your algorithm should be a polynomial in c and r. (*Hint: Think about max flow and how we could use that to solve this problem. How could we turn our input into an appropriate graph for max flow, such that finding a max flow will find a tiling?*)

Figure 1: A pixel-art dinosaur. Can you tile the the black pixels using standard 2-by-1 dominoes?

For this assignment, you must implement your algorithm in Java or Python:

- Your algorithm must be written in Python 3 (3.6.9 or below) or Java (11). We will **not** be supporting C++ in this assignment.
- You must download the appropriate wrapper code from Collab based on the language you choose: `main.py` and `tiling_dino.py` for Python, `Main.java` and `TilingDino.java` for Java.
- the `compute()` method will be given a string representation of the image as a list of Strings, which represents a $c \times r$ grid of characters defining the pixel art image you must cover. The first character in the first string will be the top-left pixel $(0, 0)$ in the image. The last character in the last string will be the bottom-right pixel $(c-1, r-1)$ in the image. Each line will contain a string where each character is either a '.' or a '#'. Any cell with '#' is "black" and must be covered by a domino, any cell with a '.' is "white" and must not be.
- Your algorithm should determine whether the image can be tiled with dominoes. If it cannot be tiled, your algorithm should return the string "impossible" as the first string in its return array. However, if it can be tiled, your algorithm should return any one valid tiling as a list/array of strings each containing four space-separated integers. Each string represents the placement of a tile by giving the coordinates (column then row, i.e. (x, y)) of the cells that domino will cover. An example input is given in Figure 2 below. Its corresponding output is given in Figure 3 below. Your dominoes may be listed in any order.

- You may write other functions in the `TilingDino.py` or `TilingDino.java` file to implement the algorithm. Do **not** modify the method definition of `compute()`.
- You *may* modify the `Main.java` or `Main.py` files to test your algorithm, but they **will not** be used during grading.
- You must submit your `TilingDino.java` or `TilingDino.py` files on Gradescope. Do **not** submit `Main.java`, `Main.py`, or any test files.
- Gradescope provides 15 submission attempts to the full autograder. Please use the "Unit D - Programming (Testing Only)" autograder to run the test case in Figure 2 and a modified version of the test case in Figure 5 provided in this PDF.
- A few other notes:
  - You may use the *networkx* package for Python or the *JGraphT* package for Java
    - Networkx: https://networkx.org/
      `pip3 install networkx` has v2.5.1 for python 3
    - JGraphT: https://jgrapht.org/
      We will use version 1.5.1
  - Your code will be run as:
    `python3 Main.py` for Python,
    or `javac *.java && java Main` for Java.
  - You may upload multiple Java files if you need additional classes, but do not assign packages to the files.

```
...#...
..###..
..###..
...#...
```

Figure 2: An example input. You should exactly cover all of the cells with "#" with dominos. Your output to this example should appear as in Figure 3. The return from your method/function should be the list/array of space-separated strings.

```
3 0 3 1
2 1 2 2
4 1 4 2
3 2 3 3
```

Figure 3: The solution for the graph given in Figure 2. The top line, "3 0 3 1" refers to placing a domino at the very top of the image covering the 0-th row 3-rd column and 1-st row 3-rd column.

```
.#...#.#...#..#####.
.#...#.#...#..#...#.
.#...#.#...#..#...#.
.#...#.#...#..#####.
.#...#..#.#...#...#.
.#...#..###...#...#.
..###....#....#...#.
```

(a) Input Strings

```
impossible
```

(b) Output Strings

Figure 4: Input and output for *test1.txt*.

```
.#...#.#...#..####..
.#...#.#...#..##.##.
.#...#.#...#..#...#.
.#...#.#...#..#.###.
.#...#..#.#...#...#.
.#####..###...#...#.
..###....#..........
```

(a) Input Strings

```
1 0 1 1
5 0 5 1
7 0 7 1
11 0 11 1
15 0 15 1
17 0 16 0
14 1 14 0
18 1 17 1
1 2 1 3
5 2 5 3
7 2 7 3
11 2 11 3
14 3 14 2
16 3 17 3
18 3 18 2
1 4 1 5
5 4 5 5
2 5 2 6
4 5 4 6
8 5 8 4
10 5 10 4
14 5 14 4
18 5 18 4
3 6 3 5
9 6 9 5
```

(b) Output Strings

Figure 5: Input and output for *test2.txt*.